HSA Overview

GREG STONER

HSA FOUNDATION'S INITIAL FOCUS

Bring Accelerators forward as a first class processor

- Unified address space across all processors
- Operates in pageable system memory
- Full memory coherency between the CPU and GPU
- Fully defined relaxed consistency memory model
- User mode dispatch/scheduling
- Eliminate drivers from the dispatch path
- QOS through pre-emption and context switching

Attract Mainstream programmers

- Support broader set of languages beyond Traditional GP-GPU Langs
- Support for Task Parallel Runtimes & Nested Data Parallel
- Rich debugging and performance analysis support

Create a platform architecture for all accelerators

- Focused on the APU/SOC

HSA FOUNDATION MEMBERSHIP – JUNE 2013

Founders

Promoters

Supporters

Contributors

Academic

Associates

DELIVERED VIA ROYALTY FREE STANDARDS

- Royalty Free IP, Specifications and API's.

- Three primary specifications are
  - HSA Platform System Architecture Specification
    - Focus on hardware requirements and low level system software
    - Support Small Mode (32bit) and Large Mode (64bit)
  - HSA Programmer Reference Manuel
    - Definition of HSAIL Virtual ISA
    - Binary format (BRIG)
    - Compiler writers guide and Libraries developer guide
  - HSA System Runtime Specification

AMD'S OPEN SOURCE COMMITMENT TO HSA

- We will open source our Linux execution and compilation stack
  - Jump start the ecosystem
  - Allow a single shared implementation where appropriate
  - Enable university research in all areas

Component Name	AMD Specific	Rationale
HSA Bolt Library	No	Enable understanding and debug
HSAIL Code Generator	No	Enable research
LLVM Contributions	No	Industry and academic collaboration
HSA Assembler	No	Enable understanding and debug
HSA Runtime	No	Standardize on a single runtime
HSA Finalizer	Yes	Enable research and debug
HSA Kernel Driver	Yes	For inclusion in linux distros

WHAT ARE THE PROBLEMS WE ARE TRYING TO SOLVE

- The SOC are quickly following into the same many CPU core bottlenecks of the PC.
  - To move beyond this we need to look at right processor(s) and/or execution device for given workload at reasonable power
- While addressing the core issues of
  - Easier to program
  - Easier to optimize
  - Easier to load balance
  - High performance
  - Lower power

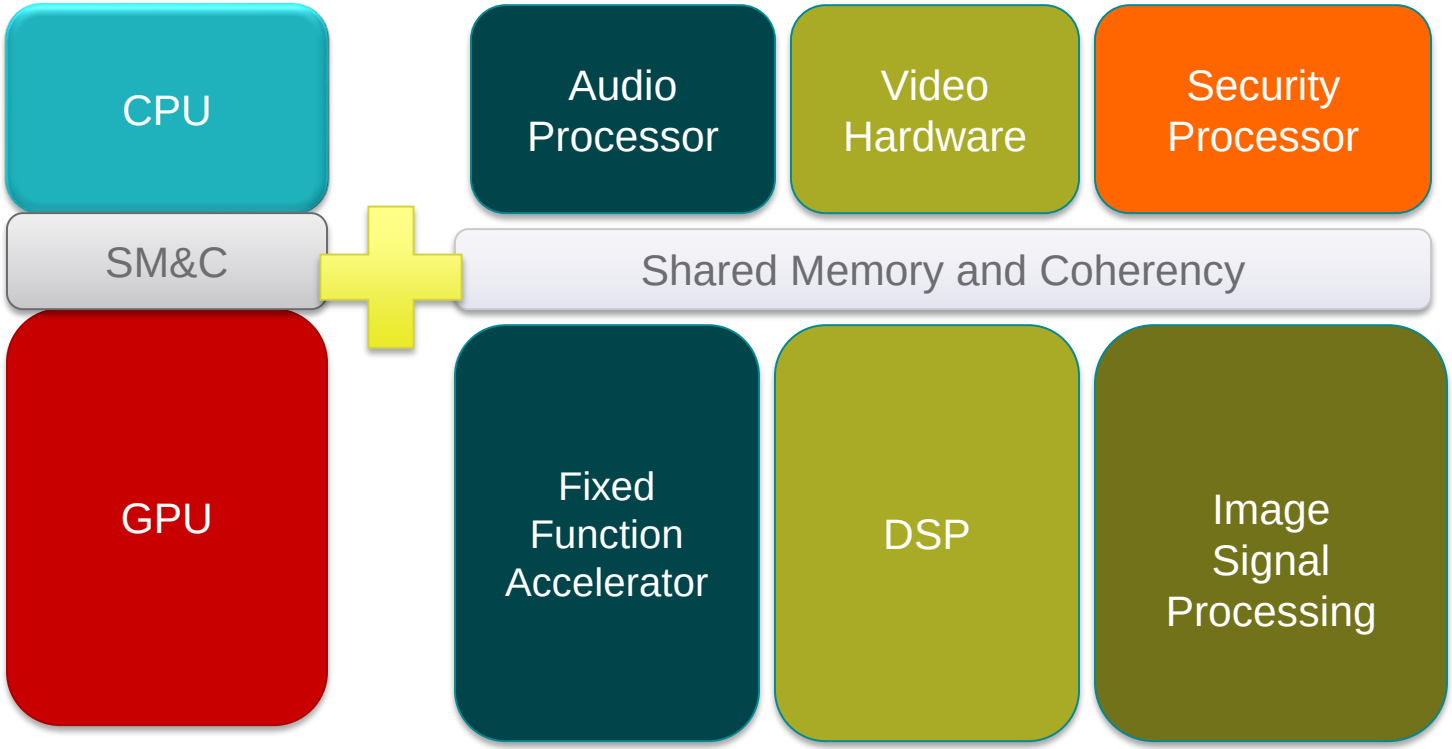
HSA TAKING PLATFORM TO PROGRAMMERS

- Balance between CPU and GPU for performance and power efficiency
- Make GPUs accessible to wider audience of programmers
  - Programming models close to today's CPU programming models
  - Enabling more advanced language features on GPU
  - Shared virtual memory enables complex pointer-containing data structures (lists, trees, etc.) and hence more applications on GPU
  - Kernel can enqueue work to any other device in the system (e.g. GPU->GPU, GPU->CPU)
    - Enabling task-graph style algorithms, Ray-Tracing, etc
- Clearly defined HSA memory model enables effective reasoning for parallel programming
- HSA provides a compatible architecture across a wide range of programming models and HW implementations.

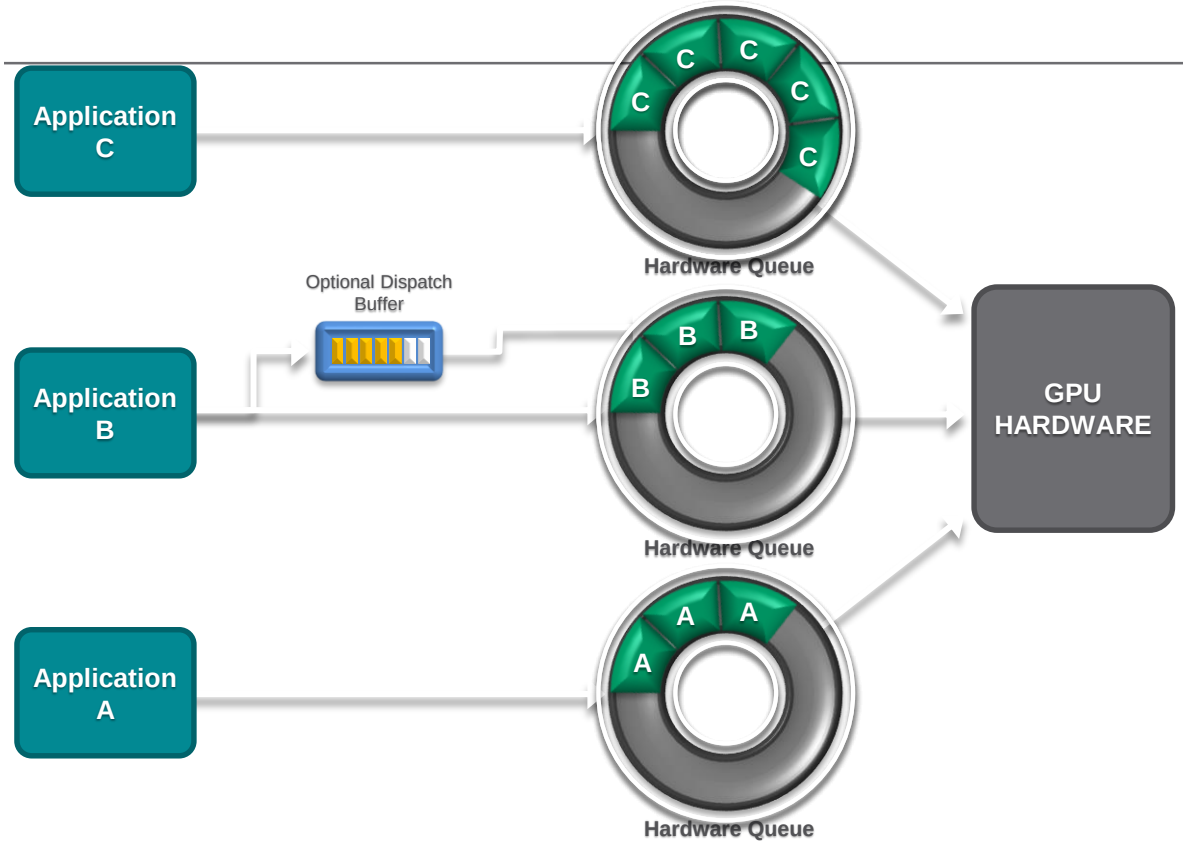
Design criteria for HSA platform infrastructure

- HSA is defined through HW requirements that enforce a set of HW compliance criteria SW can depend on
  - Comprehensive Memory model (well-defined visibility and ordering rules for transactions)
  - Shared Virtual Memory (identical page table walk between TCU and LCU VM, HW access enforcement)
  - Cache Coherency Domains
  - Memory-based signaling and synchronization
  - User Mode Queues, Architected Queuing Language (AQL)
  - Preemption / Quality of Service
  - Error Reporting
  - Syscall infrastructure (TCU can dispatch operations to LCU to call general OS APIs)
  - Hardware Debug (TCU)
  - Architected Topology Discovery
- Thin System SW Layer enables HW features for use by an application runtime
  - Mainly responsible for TCU HW init, access enforcement and resource management
  - Provides a consistent, dependable feature set for application layer through SW primitives

HSA IS DESIGNED TO GO BEYOND THE GPU



HSA COMMAND AND DISPATCH FLOW



SW view:

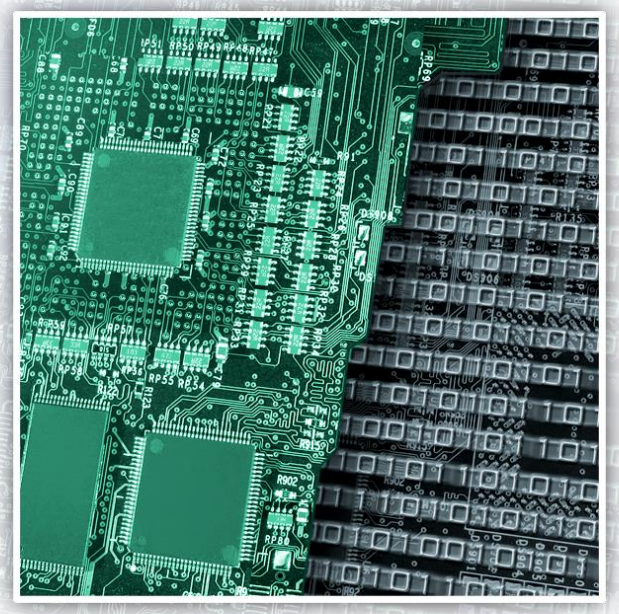
- User-mode dispatches to HW
- No KMD overhead
- Low dispatch times
- CPU & GPU dispatch APIs

HW view:

- HW / microcode controlled
- HW scheduling
- Architected Queuing Language (AQL)
- HW-managed protection

HSA MEMORY MODEL

- Defined to be compatible with C++11, Java and .NET Memory Models
- Relaxed consistency memory model for parallel compute performance
- Loads and stores can be re-ordered by the finalizer
- Visibility controlled by:
  - Load.Acquire
  - Store.Release
  - Barriers



HSAIL

- HSAIL is the intermediate language for parallel compute in HSA
  - Generated by a high level compiler (LLVM, gcc, Java VM, etc)
  - Compiled down to GPU ISA or other parallel processor ISA by an IHV Finalizer
  - Finalizer may execute at run time, install time or build time, depending on platform type
- HSAIL is a low level instruction set designed for parallel compute in a shared virtual memory environment. HSAIL is SIMT in form and does not dictate hardware microarchitecture
- HSAIL is designed for fast compile time, moving most optimizations to HL compiler
  - Limited register set avoids full register allocation in finalizer
- HSAIL is at the same level as PTX: an intermediate assembly or Virtual Machine Target
- Represented as bit-code in in a Brig file format with support late binding of libraries.

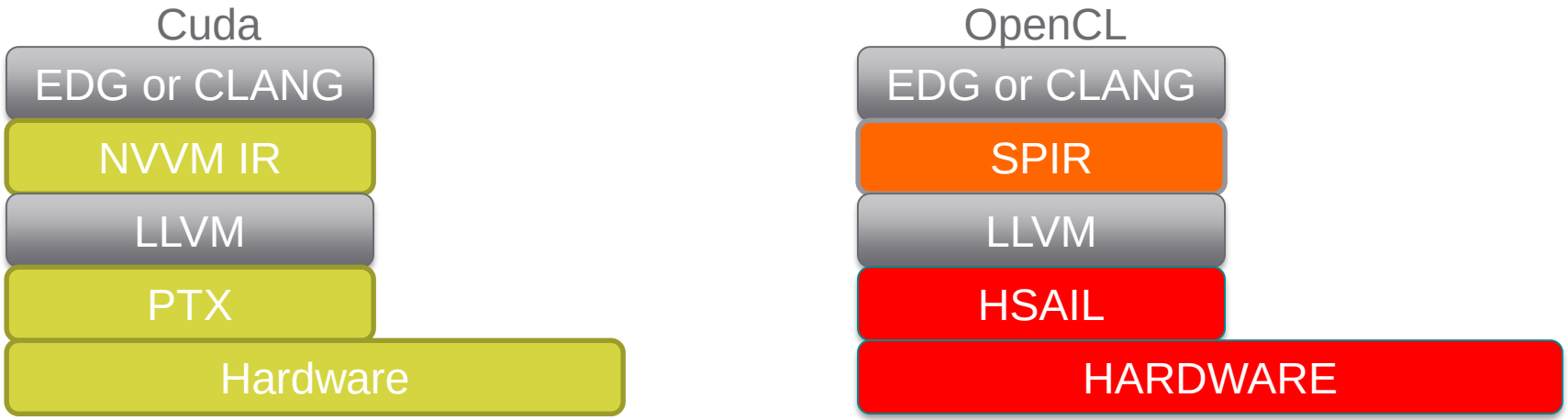
HSA Security

- With HSA, GPU operates in the same security infrastructure as the CPU
  - User and privileged memory
  - Read, write and execute protections by page table entry
- Internally, the GPU partitions functionality by privilege level
  - User mode compute queues can only run HQL packets
  - User mode graphics command buffers cannot write privileged registers
- HSA supports fixed time context switching, which is resistant to Denial of Service attacks
  - Today's GPUs are vulnerable to denial of service attacks
    - Long or infinite shader programs
    - Full GPU reset required to restore service
  - With HSA, fair scheduling and context switching ensures a responsive system

OPENCL™ AND HSA

- HSA is an optimized platform architecture, which will run OpenCL™ very well
  - Not an alternative to OpenCL™
  - Focused on the hardware platform more than API
  - Ready to support many more languages than C/C++
- OpenCL™ on HSA will benefit from
  - Avoidance of wasteful copies
  - Low latency dispatch
  - Improved memory model
  - Virtual function calls
  - Flexible control flow
  - Exception generation and handling
  - Device and platform atomics
  - Pointers shared between CPU and GPU

HSA BRINGS A MODERN OPEN COMPILATION FOUNDATION



- This bring about fully competitive rich complete compilation stack architecture for the creation of a broader set of GPU Computing tools, languages and libraries.
  - HSAIL Support LLVM and other compilers – GCC Java VM

LOOKING BEYOND GPU BASED LANGUAGES

- Dynamic Language are now one of the biggest areas that that we need rich foundation to allow for exploration of heterogeneous parallel runtimes
- Also we need a foundation that goes beyond LLVM based compilation in this environment, since many have their compilation foundation like Java/Scala, JavaScript, Dart, etc. -
- See Project Sumatra - http://openjdk.java.net/projects/sumatra/ Formal Project for GPU Acceleration for Java in OpenJDK
- We also see opportunities LLVM based environment to embrace other standards based languages like OpenMP, Fortran, GO, Haskell, and DSL's like Halide, Julia, and many other.

TOOLS ARE AVAILABLE NOW

- Tools now at GitHUB – HSA Foundation
  - libHSA Assembler and Disassembler
    - https://github.com/HSAFoundation/HSAIL-Tools
  - HSAIL Instruction Set Simulator
    - https://github.com/HSAFoundation/HSAIL-Instruction-Set-Simulator
  - HSA ISS Loader Library for Java and C++ for creation and dspatch HSAIL Kernals
    - https://github.com/HSAFoundation/Okra-Interface-to-HSAIL-Simulator
- Soon LLVM Compilation stack which outputs HSAIL and BRIG
- Will be bring C++ AMP CLANG front-end as well

GET THE PUBLIC SPEC AT

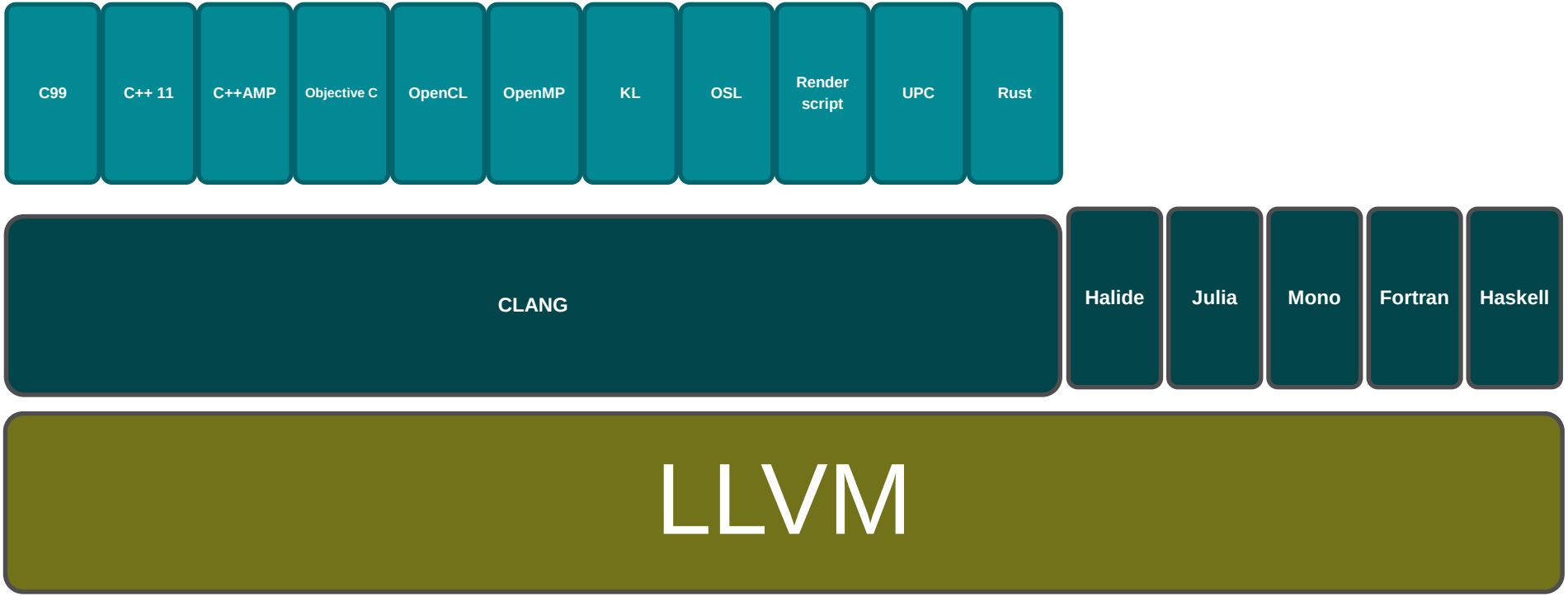
- ***HSA Programmer's Reference Manual: HSAIL Virtual ISA and Programming Model, Compiler Writer's Guide, and Object Format (BRIG)***
  - http://hsafoundation.com/standards/
  - https://hsafoundation.box.com/s/m6mrsjv8b7r50kqeyyal

BACKUP SLIDES

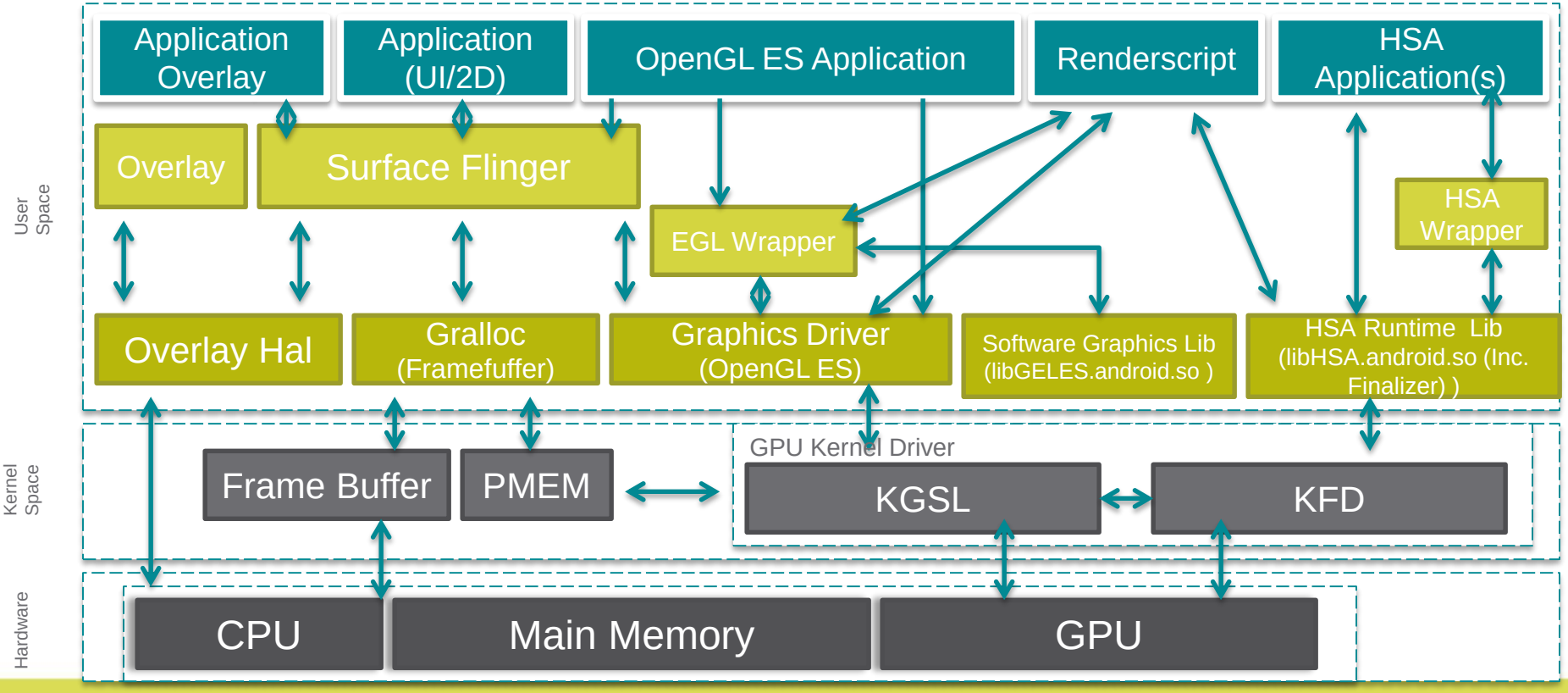
HSA Foundation System Architecture Goals

- Focus of the current “HSA System Architecture” Requirements is on defining a consistent and simple hardware operating model for use in app software, little SW involvement in performance-critical paths
  - E.g. dispatch processing can be issued from either GPU or CPU within the process context
  - The requirements describe in detail what needs to be implemented in HW for it to work, not how it needs to be implemented
  - Definition is done in a vendor-architecture neutral way -> “Big tent” approach
  - Allowing differentiation through innovation while providing a reliable baseline for software to operate on
  - Many differentiated HW architectures can map to the programming model with minor modifications
- But its goal is not (yet) to define a unified HW model (e.g. at register level)
  - All important control and prioritization mechanisms are defined, but implementation and access may differ across vendors and is covered in their system software layer
  - It is expected that a common model will be established over time, either through architected platform mechanisms (e.g. ACPI) or architected HW controls accessible to system software
- Strong system software representation helps drive a common model for HSA virtualization

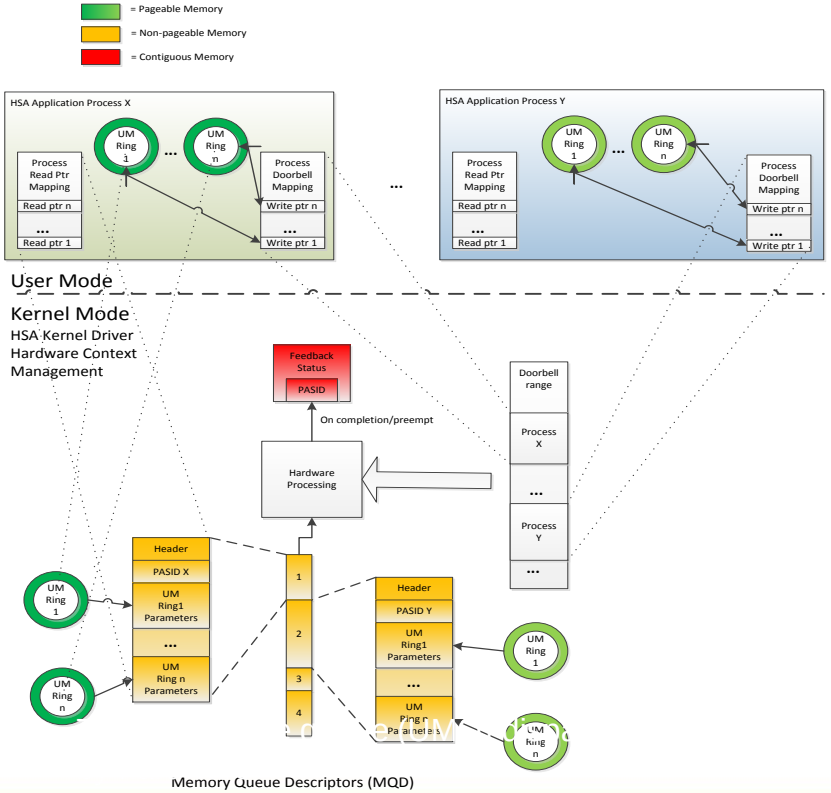
OPPORTUNITIES WITH LLVM BASED COMPILATION

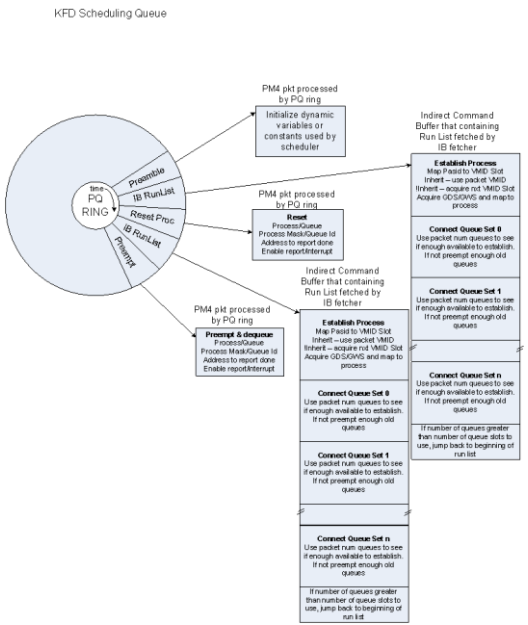


POTENTIAL ANDROID HSA STACK DIAGRAM



HSA WORKLOAD SUBMISSION PROCESSING (EXAMPLE, AMD IMPLEMENTATION)



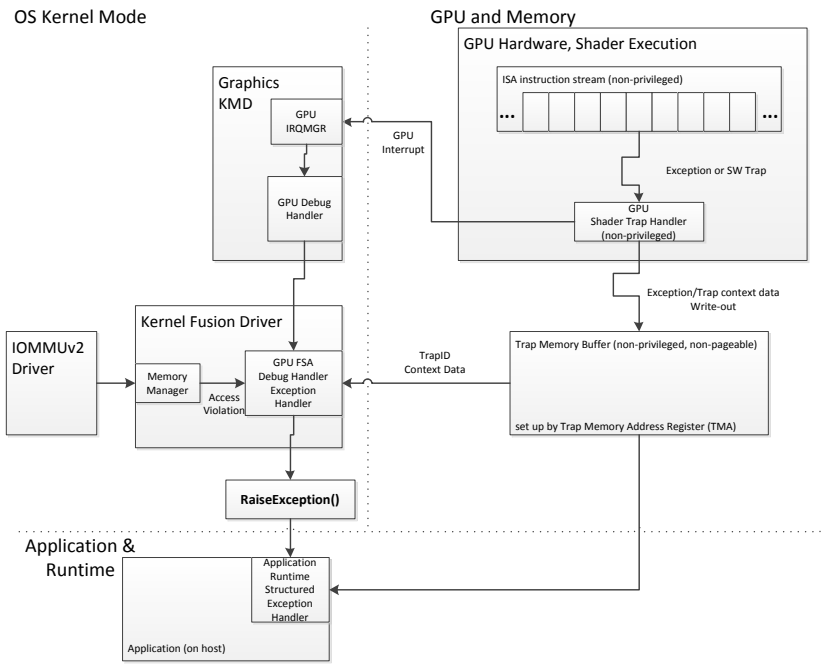


How is the HSA design accommodating virtualization?

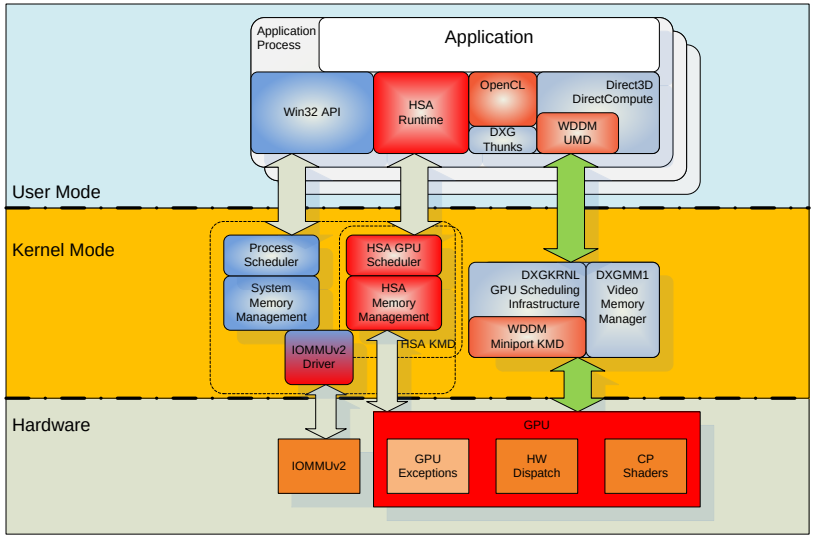
- Programming model does leverage few, simple paradigms (queues, syncvar's, events) for its operation
- All resources that are necessary to queue and dispatch workload can be expressed as memory regions
  - Privileged level SW enforces prioritization, scheduling and access control through HW mechanisms
  - The same principles can be applied to virtualize the guest OS HSA kernel resources in HV
- Workload Dispatch, prioritization/scheduling and resource management are strictly separated in the programming model
- Communication between queues and between TCU & LCU occurs via negotiated memory structures ("syncvars") within the process address space
  - The semantics are defined mostly via software, with little or no HW dependency
  - System SW uses the same mechanism to communicate events with application process software
- Software can use system "event objects" that indicate a status change in HW requiring attention
  - These are triggered by TCU interrupts and usually processed by system software on LCU
  - But the state itself is in the "syncvar" and can be processed by all HSA peers within the process

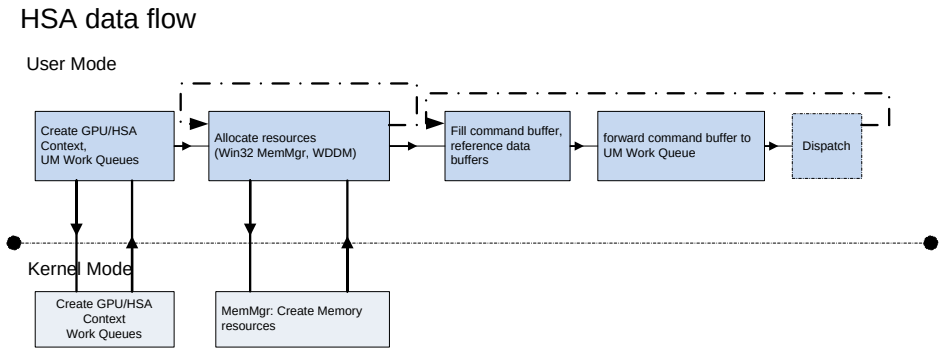
HSA EXCEPTION HANDLING (EXAMPLE, AMD IMPLEMENTATION)

- non-privileged TCU code execution causes an abnormal condition that requires attention
- Non-privileged TCU “trap handler” is invoked that classifies the condition according to policy
- If the policy requires attention of application/runtime, trap handler writes a status to non-privileged Trap Memory Buffer
- Trap handler triggers TCU interrupt, privileged SW identifies interrupt condition and raises a trap/system exception in the context of the application process
- Exception triggers LCU exception handler (runtime, app or OS default), condition is identified by evaluating Trap Memory Buffer and processed
- Exception processing is finished and queue processing is restarted by calling system software
- Approach is reused for Debug and Syscall events



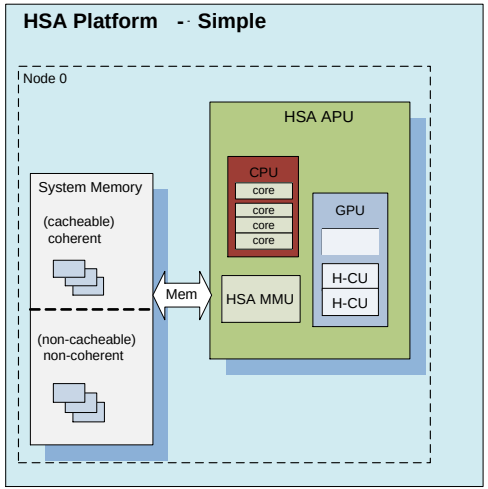
HSA COMPONENT BLOCK DIAGRAMS (AMD IMPLEMENTATION) AND DATA FLOW

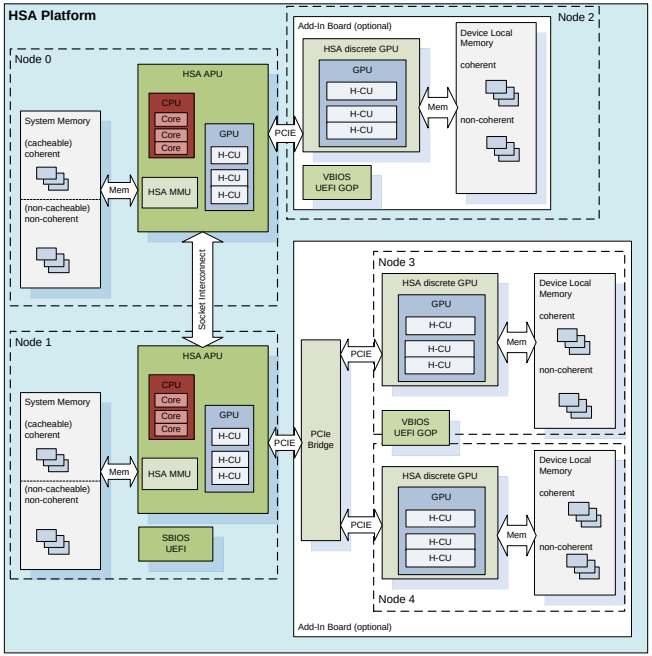




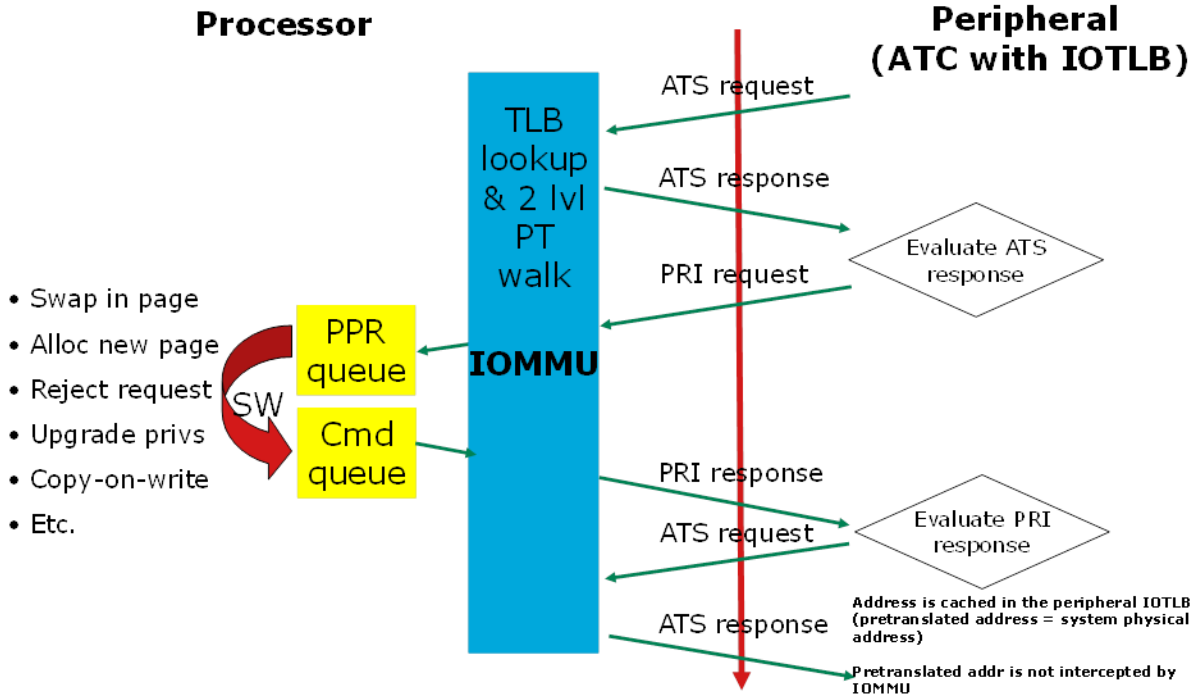
HSA Platform Topology - Example

- Strict NUMA paradigm
- HSA resources are expressed through ACPI
- System Software can discover detailed memory, cache, interconnect , LCU and TCU properties





THE IOMMUV2 OPERATION



Trademark Attribution

HSA Foundation, the HSA Foundation logo and combinations thereof are trademarks of HSA Foundation, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.